Remarks

The Applicants respectfully request reconsideration of the application in view of the following remarks.

The Examiner writes that the amendment of June 27, 2005 did not include a proper response to a Requirement for Information made in the January 2005 Office action. [Final Action, page 25.] As the Applicants noted in the previous amendment, however, the Applicants did not receive a Requirement for Information with the Office action of June 26, 2005. [*See* Amendment of June 27, 2005, page 19.] The Applicants submit herewith a response to the Request for Information provided with the final Action. The Applicants apologize for not sooner resolving with the Examiner the question of the missing Requirement for Information.

It appears that neither the arguments nor the amendments of the amendment of June 27, 2005, were considered in the final Office action of October 20, 2005 ["final Action"]. For example, the final Action did not consider new claims 37-42, nor did it consider the language added to claims 1, 6, 9, 10, 13, 16, and 18 in the amendment of June 27, 2005. With this response, the Applicants reiterate many of the arguments concerning the claims as amended in the amendment filed June 27, 2005.

Claims 1 and 4-42 are currently pending in the Application. Claims 1, 6, 9, 10, 13, 16, 18, 20, and 21 are independent. The final Action rejects claim 21 under 35 U.S.C. § 102(a) as being anticipated by "Attribute Programming with Visual C++" by Richard Grimes ["Grimes"]. Claims 1, 4-20, and 22-42 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Grimes in view of "Compiler Principles, Techniques, and Tools" by Aho et al. ["Aho"]. Additionally, claims 1, 4-10, 13-20, 22, 24-25, 27-30, 32-35 are rejected under 35 U.S.C. 103(a) as being unpatentable over Aho in view of U.S. Patent No. 5,467,472 to Williams ["Williams"]. The Applicants respectfully disagree.

I. Claims 1, 4-20 and 22-42 Are Allowable Over Grimes in View of Aho

In rejecting claims 1, 4-19, and 22-42 as being unpatentable over Grimes in view of Aho, the final Action did it consider the language added to claims 1, 6, 9, 10, 13, 16, and 18 in the amendment of June 27, 2005. The final Action also did not provide any details concerning the rejections of claims 37-42.

<u>Claims 1, 4-19 and 22-36</u>

Claims 1, 4-19 and 22-36 have been rejected under 35 U.S.C. § 103(a) as being unpatentable over Grimes in view of Aho.

Grimes and Aho, taken separately or in combination, fail to teach or suggest at least one limitation from each of independent claims 1, 6, 9, 10, 13, 16, and 18. Thus, the references fail to teach or suggest at least one limitation from each of claims 1, 4-19 and 22-36, and these claims are allowable.

Claim 1, as previously amended, recites:

> a converter module that converts the plural tokens into an intermediate representation, wherein the intermediate representation includes a symbol table and a tree that unifies representation of the programming language code and the embedded definition language information, *wherein at least some of the embedded definition language information is represented in the tree without creating new programming language code for the at least some of the embedded definition language information....*

Claim 6, as previously amended, recites:

> a converter module that converts the plural tokens into an intermediate representation comprising a symbol table and a parse tree, wherein:...
> *at least some of the interface definition language constructs are represented in the parse tree without creating new programming language constructs for the at least some of the interface definition language constructs.*

Claim 9, as previously amended, recites:

> A computer readable medium having stored thereon a data structure representing a unified interface definition language and programming language parse tree for a file having a combination of programming language code and embedded interface definition language information,...
> *wherein at least some of the embedded definition language information is represented in the parse tree without creating new programming language code for the at least some of the embedded definition language information.*

Claim 10, as previously amended, recites:

> converting the one or more input files into one or more output code files that include fragments of definition language information, ... *wherein at least some of the definition language constructs are represented in the tree without creating new programming language constructs for the at least some of the definition language constructs....*

Claim 13, as previously amended, recites:

> converting the plural tokens into an intermediate representation, wherein the converting comprises building a tree that unifies representation of the programming language code and the embedded definition language information and *the building comprises representing at least some of the embedded definition language information in the tree without creating new programming language code for the at least some of the embedded definition language information ...*

Claim 16, as previously amended, recites:

> building a tree that unifies representation of the embedded definition language information and the programming language code, *wherein the building comprises representing at least some of the embedded definition language information in the tree without creating new programming language code for the at least some of the embedded definition language information.*

Claim 18, as previously amended, recites:

> a converter module that converts the plural programming language tokens and plural definition language tokens into an intermediate representation, wherein the intermediate representation includes a tree that unifies representation of the definition language information and the programming language code, the converter module further checking for syntax errors, and *wherein at least some of the definition language information is represented in the tree without creating new programming language code for the at least some of the definition language information ....*

Grimes and Aho, taken separately or in combination, fail to teach or suggest the above-cited language from each of claims 1, 6, 9, 10, 13, 16, and 18, respectively. For the sake of illustration (and without implying any boundary or limitation on the scope of claim 1), one example of subject matter falling within claim 1 is given on page 24 of the Application:

> In Figure 7, the output module 778 directly manipulates internal compiler structures such as the symbol table 730 and the parse tree 732, creating symbols, adding to the parse-tree, etc.

Another example is given in the Application on page 26:

> While executing concurrently with the compiler, the IDL attribute provider detects (act 852) the occurrence of designated events within the compiler, for example events relat[ing] to the state of compilation .... Examining the state, the IDL attribute provider can wait until the compiler reaches a certain state, and then perform an operation when that state is reached, for example,

> requesting the compiler to modify the parse tree. The IDL attribute provider then waits for another event.
>
> The IDL attribute provider can perform different operations for different events that might occur within the compiler, and for different parameters transmitted with an IDL attribute. Among these operations are injection of statements or other program elements, possibly employing templates, and modifying or deleting code. Other operations include adding new classes, methods and variables, or modifying existing ones. Modification can include renaming and extending an object or construct.

Grimes describes injecting C++ code in response to attributes generally. [Grimes, pages 2-3.] According to Grimes:

> *When the compiler sees an attribute in your C++ code* it passes the attribute and its arguments to the attribute providers that have been registered on the system. The compiler does this by calling methods on an interface called IAttributeHandler that is implemented by the provider. So that the provider has access to the compiler, it also passes a callback interface pointer of the type ICompiler. If the provider recognizes the attribute, it can then call methods on ICompiler to tell the compiler to *add C++ code to replace the attribute.*

[Grimes, page 3.] This involves replacing an attribute with C++ code when the compiler encounters the attribute. In other words, it involves creating C++ code for the attribute. Replacing an attribute with C++ code when the compiler encounters the attribute (as in Grimes) leads away from the above-cited language of claims 1, 6, 9, 10, 13, 16, and 18, respectively.

Aho does not teach or suggest the above-cited language from claims 1, 6, 9, 10, 13, 16, 18, respectively. Aho generally describes compiler techniques and tools. The parts of Aho specifically cited by the Examiner describe, among other things, compiler organization by "front end" and "back end" [Aho, page 20], symbol tables [Aho, page 11], and parse trees [Aho, pages 6 and 40-48], but do not address working with both programming language and definition language. These parts of Aho additionally do not address the direct manipulation of parse trees and symbol tables in response to IDL attributes.

The Examiner writes:

> Aho demonstrated that it was known at the time of invention to develop compilers with a front end, a converter module, and a back end.... It would have been obvious to one of ordinary skill in the art at the time of invention to implement Grimes' system of C++ code and definition code with a compiler, which would generate executable code as found in Aho's teaching. This

> implementation would have been obvious because one of ordinary skill would be motivated to provide a mechanism, which would allow source code to produce meaningful executable code.

[Final Action, page 5.] Elsewhere, the Examiner writes:

> Aho demonstrated that it was known at the time of invention to utilize parse trees and symbol tables.... It would have been obvious to one of ordinary skill in the art at the time of invention to implement Grime's interface definition language / programming language compiler with symbol tables and parse trees as appropriate for compiling such a combination as found in Aho's teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to use common and well understood techniques for implementing compilers.

[Final Action, page 8.] The Applicants respectfully disagree with these statements from the Examiner. Even if, for the sake of argument, the compiler of Grimes were to be implemented with a symbol table and parse tree as described in Aho (or with any other symbol table and parse tree, as those terms are broadly understood in the art), the symbol table and parse tree would be used for processing C++ code but not for also processing IDL. Grimes makes this clear because it describes (1) processing IDL attributes and the generated IDL apart from C++ code during compilation and (2) replacing an attribute with C++ code when the attribute is encountered by the compiler. Thus, even for a file with C++ code and IDL attributes, Grimes suggests that any compilation involves only compilation of C++ code. Even if combined, the combination of Grimes and Aho does not teach or suggest the above-cited language of claims 1, 6, 9, 10, 13, 16, and 18, respectively.

For at least these reasons, claims 1, 6, 9, 10, 13, 16, and 18 should be allowable. In view of the foregoing discussion, the Applicants will not belabor the merits of the separate patentability of dependent claims 4, 5, 7, 8, 11, 12, 14, 15, 17, 19, and 22-36.

Claim 20

Claim 20 has been rejected under 35 U.S.C. § 103(a) as being unpatentable over Grimes in view of Aho. Grimes and Aho, taken separately or in combination, fail to teach or suggest at least one limitation from claim 20.

Claim 20 recites:

> modifying the programming language compiler to *expose the compiler state to one or more interface definition language attribute providers*; and
>
> modifying the programming language compiler to allow *manipulation of the symbol table and the parse tree by the one or more interface definition language attribute providers based upon the semantics of the embedded interface definition language information,* wherein the parse tree unifies representation of the interface definition language information and the programming language source code.

Grimes, describes communication between the compiler and the attribute provider. [Grimes, pages 2-3.] According to Grimes:

> When the compiler sees an attribute in your C++ code *it passes the attribute and its arguments to the attribute providers that have been registered on the system.* The compiler does this *by calling methods on an interface called IAttributeHandler that is implemented by the provider.* So that the provider has access to the compiler, it also passes a callback interface pointer of the type ICompiler. If the provider recognizes the attribute, *it can then call methods on ICompiler to tell the compiler to add C++ code to replace the attribute.*

[Grimes, pages 2-3.] Thus, according to Grimes, not only does *the compiler call a registered attribute provider* when it "sees" an attribute, but the attribute provider *instructs the compiler to add code* to replace the attribute. Because the attribute provider responds by instructing that code be added, rather than responding with manipulation of any intermediate representations present in the compiler, Grimes does not teach or suggest "modifying the programming language compiler to expose the compiler state to one or more interface definition language attribute providers," as recited in claim 20. Aho also does not teach or suggest the above-cited language of claim 20. For at least these reasons, claim 20 should be allowable.

Claims 37-42

The final Action rejected dependent claims 37-42 as being unpatentable over Grimes in view of Aho, but the final Action did not provide any detail for the rejections of claims 37-42. Dependent claims 37-42 should be allowable.

II. Claim 21 Is Allowable Over Grimes

Claim 21 recites:

> embedding by the programming language compiler debugging information in a definition language output file, the definition language output file for subsequent processing by a definition language compiler, whereby the embedded debugging information associates errors raised by the definition language compiler with locations of embedded definition language constructs in the input file to facilitate debugging of the input file.

Claim 21 has been rejected under 35 U.S.C. § 102(a) as being anticipated by Grimes. However, Grimes fails to teach or suggest the above-cited language from claim 21. The final Action cites various sections of pages 2 and 3 of Grimes, claiming they map to the language recited in claim 21. The Applicants disagree with this characterization of Grimes.

On page 3, Grimes states:

> *The combination of the original and the 'injected' code is compiled to generate an .obj file* and, since this is carried out by the compiler, you do not see any of the generated code. However, in debug builds the compiler will store information about the ATL code *that was generated* in the project's PDB file, so that when you debug attributed code you can step into *the generated code.*

As this passage makes clear, at page 3, Grimes describes the usage of original and injected *C++ code* to create a .obj file. Because the .obj file is created from C++ code, and not by compilation of IDL information, the stored debug information of page 3 of Grimes would be directed toward the original or injected C++ code itself, and not any IDL information. Indeed, even if IDL information were replaced with injected C++ code, the IDL information would be lost by the time the .obj file is generated.

By contrast, the other cited passages of Grimes discuss "add[ing] attribute information to the .obj file which can be extracted with the Idlgen utility to generate IDL and then run[ing] MIDL to create the type information." [Grimes, page 3.] Applicants note that these passages do not discuss the storing of debug information (and are thus in contrast to the passage of Grimes addressed in this section above, which does discuss storing of debug information, but not of the type recited in claim 21).

For at least these reasons, Grimes does not teach or suggest each and every limitation of claim 21. Claim 21 should thus be allowable.

III. Claims 1, 4-10, 13-20, 22, 24-25, 27-30, 32-35 Are Allowable Over Aho in View of Williams

Claims 1, 4-10, 13-20, 22, 24-25, 27-30, and 32-35 have been rejected under 35 U.S.C. § 103(a) as being unpatentable over Aho in view of Williams. In rejecting these claims as being unpatentable over Aho in view of Williams, however, the final Action did it consider the language added to claims 1, 6, 9, 10, 13, 16, and 18 in the amendment of June 27, 2005. Aho and Williams, taken separately or in combination, fail to teach or suggest at least one limitation from each of independent claims 1, 6, 9, 10, 13, 16, 18 and 20. Thus, the references fail to teach or suggest at least one limitation from each of claims 1, 4-10, 13-20, 22, 24-25, 27-30, and 32-35, and these claims are allowable.

Claim 1 recites "embedded definition language information" as well as "programming language code." In fact, each of claims 6, 9, 10, 13, 16, 18, and 20 recites "definition language constructs" or "definition language information" as well as "programming language constructs" or "programming language code" or "programming language source code" recited in the respective claims. Similarly, for example, the Application, at page 2, lines 2-6 indicates:

> Conventionally, programmers use a definition language to create a specification for an object. This definition language specification defines how the object interacts with other objects. *With reference to this definition language specification, programmers then use a programming language to actually write code for the object.*

The final Action acknowledges the recited language of claim 1 is not found in Aho, but claims it is found in Williams. In the rejection of each of the other independent claims, the final Action alleges that "[t]he limitations are substantially similar to those of claim 1 and as such are rejected in the same manner." In its rejection of Claim 1, the final Action cites to column 2, lines 52-60 of Williams, which reads:

> To allow an object of an arbitrary class to be shared with a client program, interfaces are defined through which an object can be accessed without the need for the client program to have access to the class definitions at compile time. An interface is a named set of logically related function members. *In C++, an interface is an abstract class with no data members and whose virtual functions are all pure.* Thus, an interface provides a published protocol for two programs to communicate.

The Applicants note that the cited passage of Williams references "class definitions" and "a published protocol" but does not anywhere describe a "definition language." In fact, Williams at column 1, lines 52-56 describes class declarations *in C++*.

> In the C++ language, data encapsulation and inheritance are supported through the use of classes. A class is a user-defined type. A class declaration describes the data members and function members of the class.

As noted above, "definition language," as recited in claim 1, is something other than simple "programming language." Williams does not teach or suggest the "definition language" limitations recited in claims 1, 6, 9, 10, 13, 16, 18 and 20, respectively. Additionally, in the "Response to Arguments" section on pages 25 and 26, the final Action states that "[w]ith regard to Applicant's arguments over the Williams reference, the claim language states 'definition language information.' This term is not limited in any way by Applicant's specification and is broad." Notwithstanding the fact that the language "definition language information" does not identically appear in every claim rejected over Aho in view of Williams, the Applicants note that Williams does not teach or suggest the "definition language" limitations recited in claims 1, 6, 9, 10, 13, 16, 18 and 20, respectively.

For at least these reasons, Aho and Williams, taken separately or individually, do not teach or suggest at least one limitation of each of claims 1, 6, 9, 10, 13, 16, 18 and 20, respectively. These claims should be allowable. In view of the foregoing discussion, the Applicants will not belabor the merits of the separate patentability of dependent claims 4, 5, 7, 8, 14, 15, 17, 19, and 22, 24-25, 27-30, and 32-35.

Requirement for Information Under 37 C.F.R. § 1.105

The final Action provides a Requirement for Information. In response to the Requirement, the Applicants have located and produced following documents, which are included with the instant Response:

1. A copy of a two-page agenda for the Microsoft Professional Developers Conference, held October 11-15, 1998 in Denver, Colorado ("Exhibit A").

2. A copy of a 52-slide slide show entitled "C++ Language Innovations for COM and COM+" which was presented as part of the conference talk "Language Innovations for COM+ and Beyond" ("Exhibit B").

3. A 4-page declaration by inventors Paul Ringseth, Jonathan Caves, and Jason Shirk, of Microsoft Corporation, which describes release-time capabilities of the Visual C++ Technical Preview, released at the Denver PDC, as well as its technical and theoretical capabilities as they were then known to the public ("Exhibit C").

After investigation, the Applicants believe that these documents provide as complete a response to items 1, 2, 4, and 5 of the Requirement as is readily available. The Applicants have not been able to locate the material requested in item #2 of the Requirement, a copy of the Denver PDC DVD file titled "Visual C++ Technical Preview." The Applicants have also not been able to locate a transcript for a conference talk entitled, "Language Innovations for COM+ and Beyond," as requested in item 3 of the Requirement, or for any other conference talk at the Denver PDC. The Applicants believe that document #2 provides information concerning the subject matter discussed at the conference talk.

In response to the non-numbered requests for the names of products and services which incorporated either the claimed subject matter or the "disclosed prior art" of Grimes, the Applicants do not know of any products or services which incorporated the claimed subject matter prior to July 6, 2000, which is the filing date of the instant application. As for the "disclosed prior art" of Grimes, the Applicants do not know of any products or services which incorporated Richard Grimes' analysis as stated in the Grimes article.

If the Examiner seeks additional information on any of these points, the Applicants respectfully request the Examiner call the undersigned attorney.

Conclusion

Claims 1 and 4-42 should be allowable. Such action is respectfully requested.

Request for Interview

In view of the preceding remarks, the Applicants believe the application to be allowable. If any issues remain, however, the Examiner is formally requested to contact the undersigned attorney at (503) 226-7391 prior to issuance of the next communication in order to arrange a telephonic interview. This request is being submitted under MPEP § 713.01, which indicates that an interview may be arranged in advance by a written request.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

By [illegible]
Kyle B. Rinehart
Registration No. 47,027

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
Facsimile: (503) 228-9446

Microsoft Professional Developers Conference
October 11 – 15, 1998
Colorado Convention Center

Early Registration: Saturday, Oct 10, 5:00-9:00pm

Pre-conference: Sunday, Oct 11

6:30a – 9:30p	Registration	Lobby B
6:30 – 8:30a	Continental Breakfast	Hall A
8:30 – 12:30p	The COM Programming Model – Don Box	Ballroom 1
	Windows NT Internals – David Soloman	Currigan 3
	Developing Distributed Applications using MTS – Steven Gray	Currigan 2
12:30-1:30p	Box lunch	Lobby B, Currigan
1:30-5:30p	Distributed Programming with DCOM and MTS – Don Box	Ballroom 1
	Windows NT5 – New Architecture and APIs – Jeffrey Richter	Currigan 3
	Windows NT Management, Directory & Security Services – Margaret Johnson & Dan Plastina	Currigan 2
5:00 – 9:30p	Opening reception / Trade Show / HOT	

Day 1: Monday, Oct 12

6:30a – 8:30a	Registration, Continental Breakfast	Lobby B / Hall A
8:30 – 9:00a	Welcome – Tod Nielsen	Hall C
9:00 – 10:00a	Bill Gates keynote	Hall C
10:00 – 11:00a	Evolving the Windows Platform into the Next Decade – David Vaskevitch	Hall C
11:00 – 11:30a	break	
11:30 – 12:30	Tools for Building Windows DNA-based Applications – Paul Gross	Hall C
12:00 – 3:00p	Trade Show	
12:30 – 2:30p	Lunch / HOT	Hall A
2:30 – 3:45p	Breakout Sessions (1)	
3:45 – 4:15p	Break	
4:15 – 5:30p	Breakout Sessions (2)	
5:30 – 8:30p	HOT	
5:30 – 9:30p	Trade Show / Reception	Ballroom 2/3
9:30 – 11:30p	Late Night at the Movies (TBD)	Performing Arts Center

Day 2: Tuesday, Oct 13

7:00a – 9:00p	Registration	Lobby B
6:30 – 8:00	Continental Breakfast	Hall A
8:00 – 8:30a	Top 10 – Tod Nielsen	Hall C
8:30 – 10:30a	Building Windows DNA-based Applications – Chris Jones, J Allard, Dave Reed	Hall C
10:30 – 11:00a	Break	
11:00 – 12:15p	Breakout Session (3)	
12:15 – 2:15p	Lunch/HOT	
11:45 – 2:45p	Trade Show	Ballroom 2/3
2:15 – 3:30p	Breakout Session (4)	Hall A
3:30 – 4:00p	Break	
4:00 – 5:15	Breakout Session (5)	
5:15 – 5:30p	Break	
5:30 – 6:45p	Breakout Session (6)	
6:30 – 9:30p	Trade Show / Reception / ATE	

Day 3: Wednesday, Oct 14

7:00a – 8:00p	Registration	Lobby B
6:30 – 8:00a	Continental Breakfast	Hall A
8:00 – 8:30a	Top 10 – Tod Nielsen	Hall C
8:30 – 9:30a	Applications Product Strategy – Bob Muglia	Hall C
9:30 – 10:00a	Break	
10:00 – 11:15a	Breakout Sessions (7)	
11:15 – 11:30a	Break	
11:30 – 12:45p	Breakout Sessions (8)	
12:15 – 3:15p	Trade Show	Ballroom 2/3
12:45 – 2:45p	Lunch / HOT	Hall A
2:45 – 4:00p	Breakout Sessions (9)	
4:00 – 4:30	Break	
4:30 – 5:45p	Breakout Sessions (10)	
6:00 – 9:30p	Dinner / Penn & Teller	National Western Complex

Day 4: Thursday, Oct 15

7:00a – 5:00p	Registration	Lobby B
6:30 – 8:00a	Continental Breakfast	Hall A
8:00 – 8:30a	Call to Action/Wrap Up – Tod Nielsen	Hall C
8:30 – 9:30a	Windows CE: Building a Foundation for Pervasive Computing – Harel Kodesh	Hall C
9:30 – 10:30a	Windows NT Q&A – Jim Allchin	
10:30 – 11:00a	Break	
11:00 – 12:15p	Breakout Sessions (11)	
12:15 – 1:00p	Pick up Boxed Lunch	
1:00 – 2:15p	Breakout Sessions (12)	
2:15 – 2:45p	Break	
2:45 – 4:00	Breakout Sessions (13)	
4:00 – 4:15p	Break	
4:15 – 5:30p	Breakout Sessions (14)	
5:30p	End of Conference	

C++ Language Innovations For COM And COM+

Jonathan Caves
Roland Fernandez
Visual C++ Development
Microsoft Corporation

EXHIBIT B
Page 1 of 52

Introduction

- WinDNA, COM, and n-tier are the future
- Performance will still be critical
- Visual C++® is the tool for performance
- But. . .COM programming in Visual C++ can be difficult

EXHIBIT B
Page 2 of 52

Where We Are Today

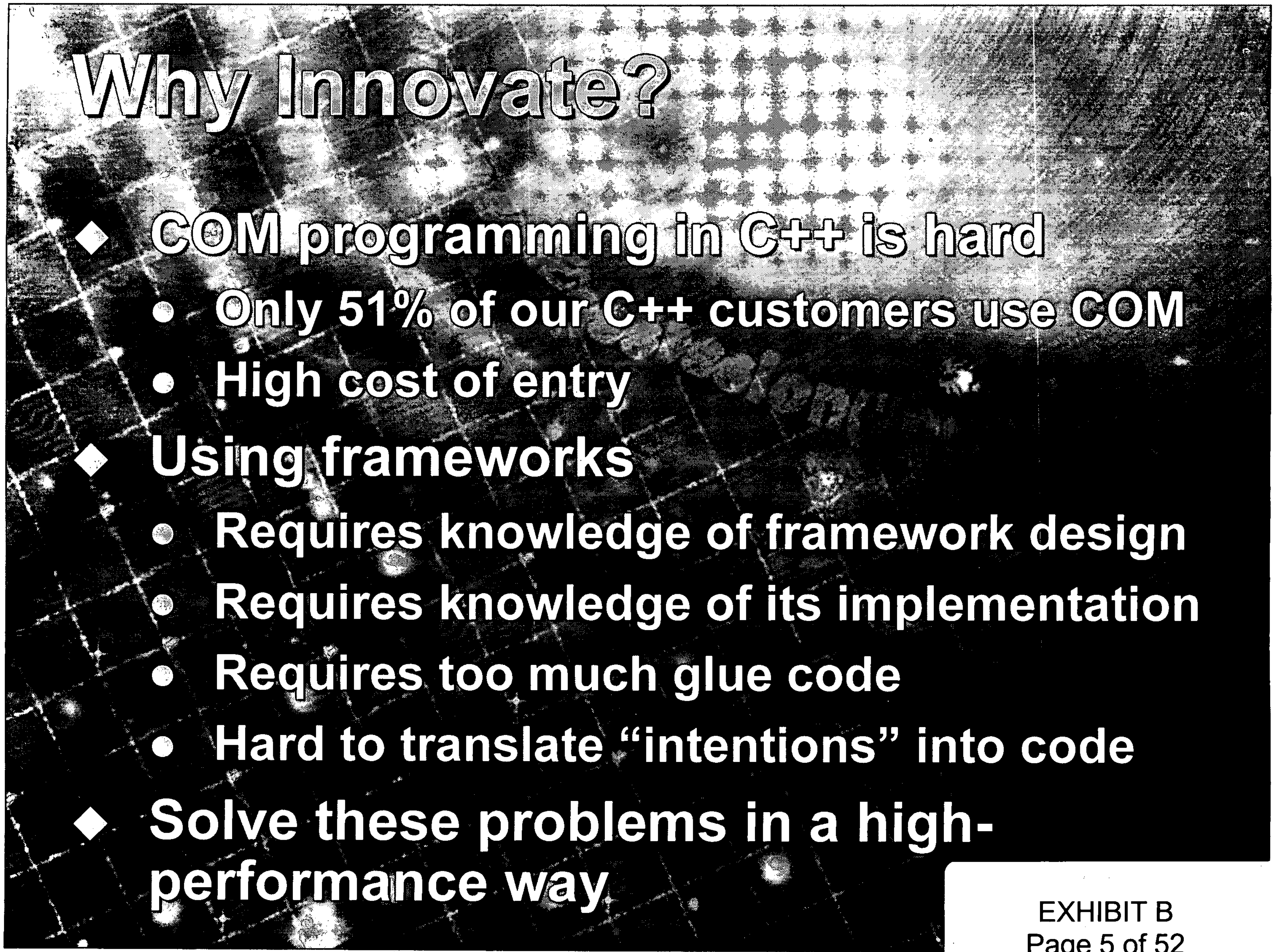
Demo

EXHIBIT B
Page 3 of 52

Language Innovation

- Why Innovate?
- Innovations
  - Attributes
  - The 'interface' keyword
  - The 'property' attribute
- Putting it all together

EXHIBIT B



Why Innovate?

- COM programming in C++ is hard
  - Only 51% of our C++ customers use COM
  - High cost of entry
- Using frameworks
  - Requires knowledge of framework design
  - Requires knowledge of its implementation
  - Requires too much glue code
  - Hard to translate "intentions" into code
- Solve these problems in a high-performance way

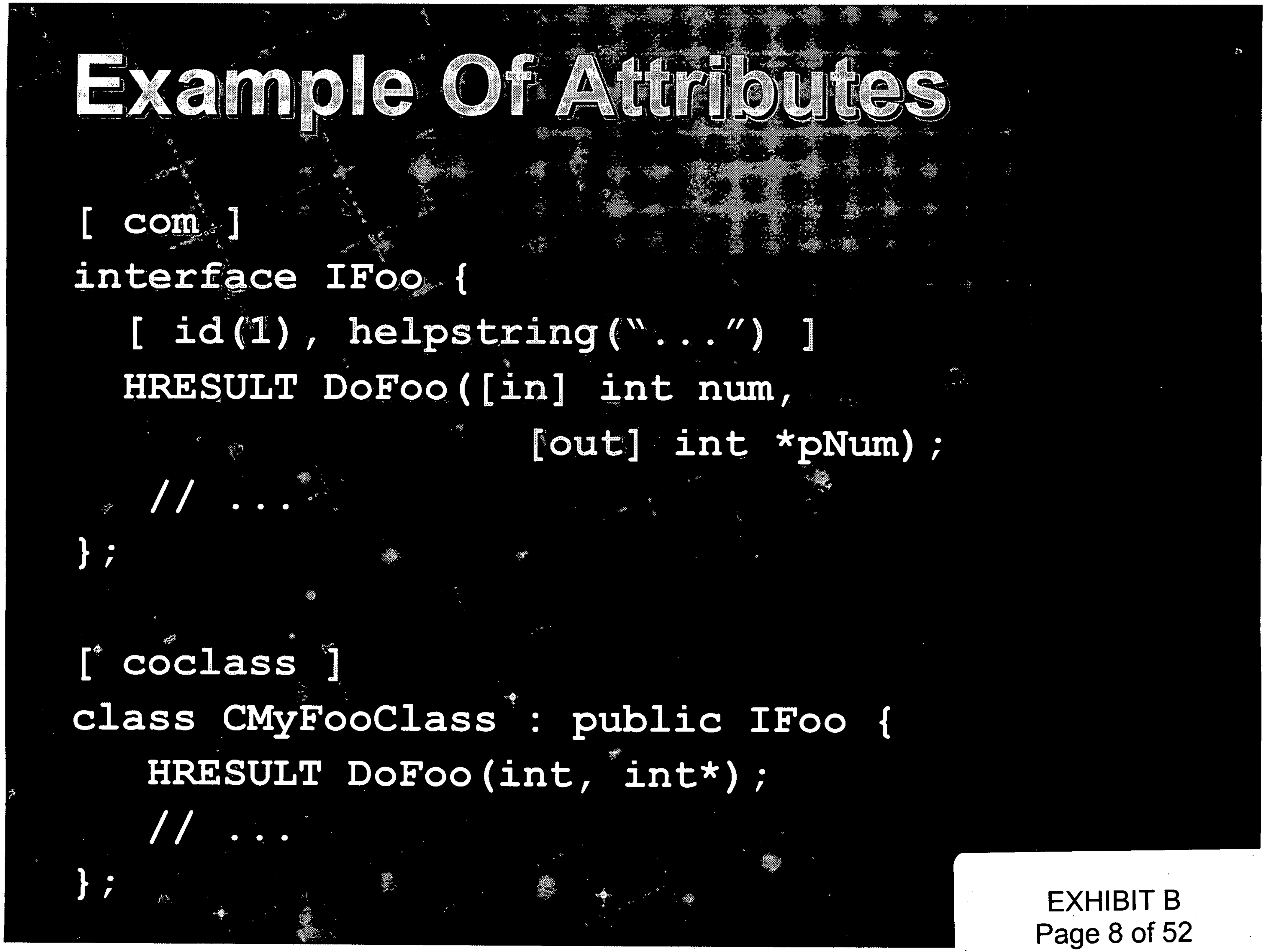
EXHIBIT B
Page 5 of 52

Why Innovate?

- Simplify programming model and access to frameworks
  - Declarative (maps concepts to implementation)
  - Source annotations that guide implementation
- Support for interface based programming

What Is Attributed Programming?

- Declarative instructions that guide the implementation
- Hints to the compilation process
  - In-place or multi-site code transformations
  - Insert information into meta-data



Example Of Attributes

```
[ com ]
interface IFoo {
  [ id(1), helpstring("...") ]
  HRESULT DoFoo([in] int num,
                [out] int *pNum);
   // ...
};

[ coclass ]
class CMyFooClass : public IFoo {
   HRESULT DoFoo(int, int*);
   // ...
};
```

Design Goals Of Attribute Programming

- Encapsulate domain concepts
- Limit C++ language extensions
- Don't change C++ Semantics
- Maintain compatibility with existing C++ code

Benefits Of Attribute Programming

- Decreases domain knowledge
- Lowers cost of entry
- User scalable solution
- Generates standard C++
- No C++ language compromise
  - Full control and power available

Language Innovation

- Why Innovate?
- **Innovations**
  - **Attributes**
  - The 'interface' Keyword
  - The 'property' Attribute
- Putting it all together

Attribute Syntax

- Attributes can be applied to any declaration

```
[ attr ]
class B {
public:
  [ attr ] void mf() { }
  [ attr ] int m_data;
};

[ attr ] void f() { }
```

EXHIBIT B
Page 12 of 52

Attribute Syntax

- The attributes are always applied to the declaration not to the type or the instance

```
[ attr ]                [ attr ]
const class X {         unsigned int f() {
   // ...                  // ...
} x;                    }
```

Attribute Syntax

- The attributes are always applied to the declaration not to the type or the instance

```
[ attr ]                  [ attr ]
const class X {           unsigned int f() {
   // ...                    // ...
} x;                      }
```

Attribute Syntax

- The syntax is based upon IDL
- Several different styles of attributes

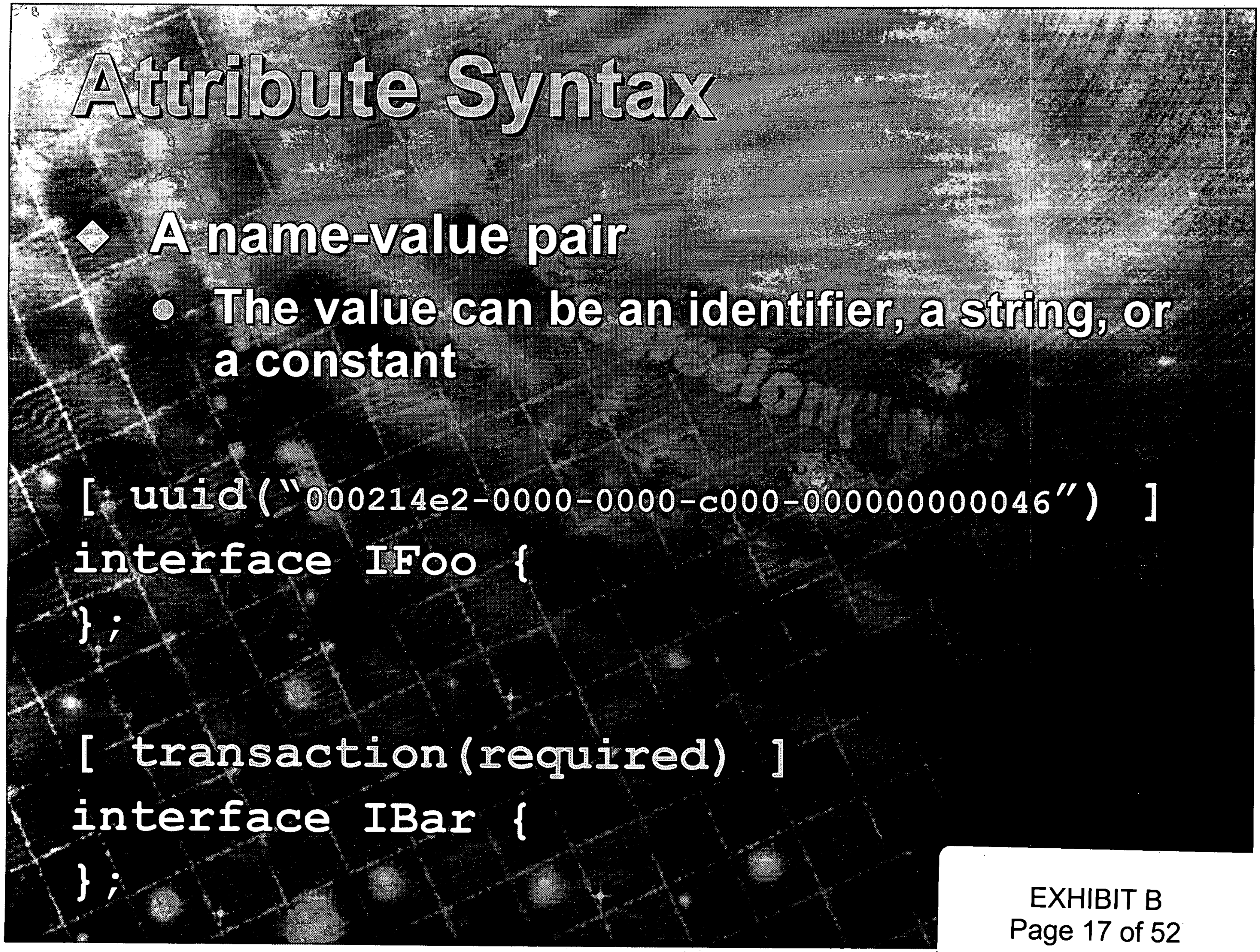
EXHIBIT B
Page 15 of 52

Attribute Syntax

- Just a simple name

```
[ com ]
interface IFoo {
   HRESULT mf([in] int a);
};

[ coclass ]
class CFoo : public IFoo {
   // ...
};
```

EXHIBIT B
Page 16 of 52



Attribute Syntax

- A name-value pair
  - The value can be an identifier, a string, or a constant

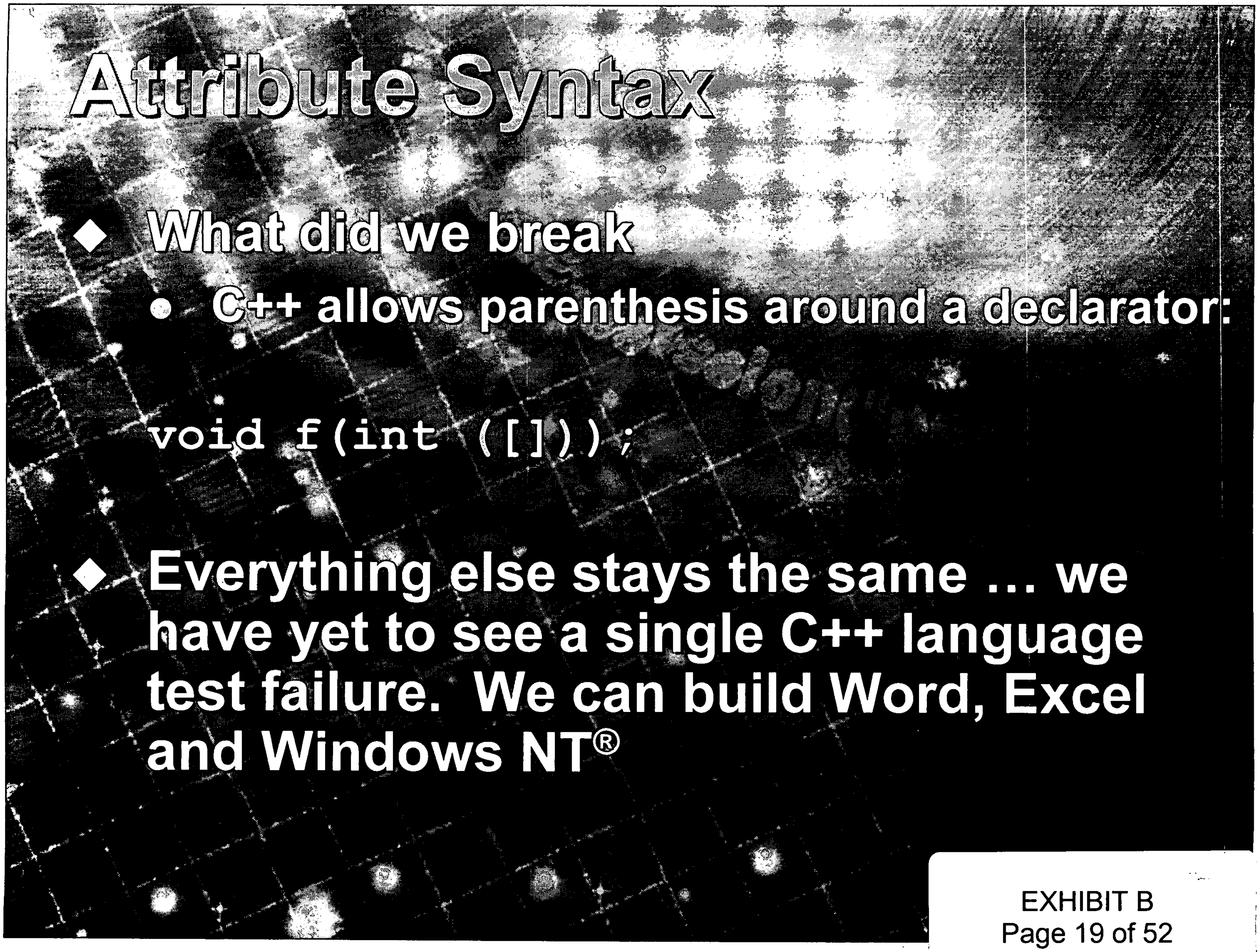
```
[ uuid("000214e2-0000-0000-c000-000000000046") ]
interface IFoo {
};

[ transaction(required) ]
interface IBar {
};
```

EXHIBIT B
Page 17 of 52

Attribute Syntax

- A name and a set of properties
- Where properties can either be simple names or a name-value pair

```
[ module(dll, name="MyControl") ];
```



Attribute Syntax

- What did we break
  - C++ allows parenthesis around a declarator:

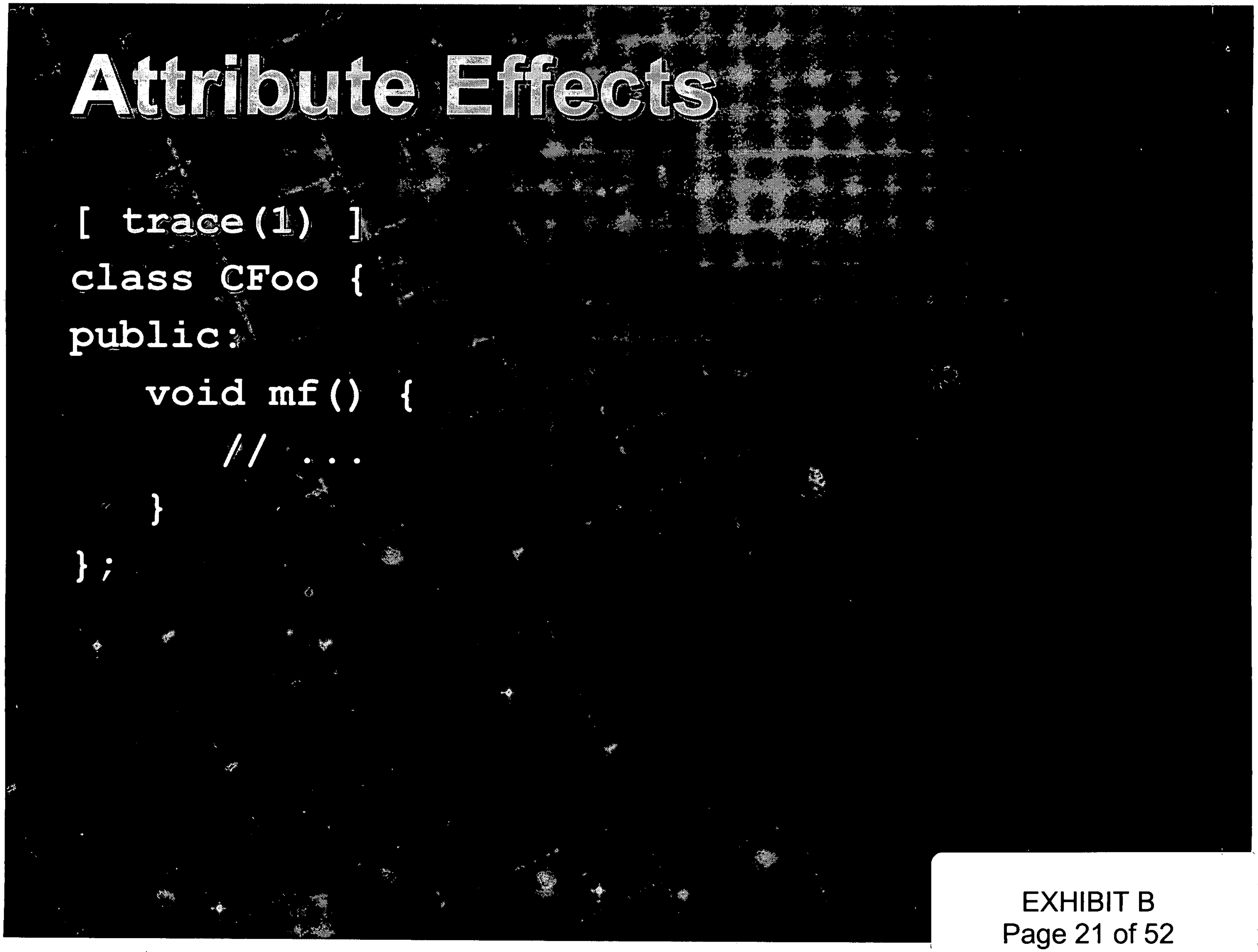
```
void f(int ([]));
```

- Everything else stays the same … we have yet to see a single C++ language test failure. We can build Word, Excel and Windows NT®

EXHIBIT B
Page 19 of 52

Attribute Effects

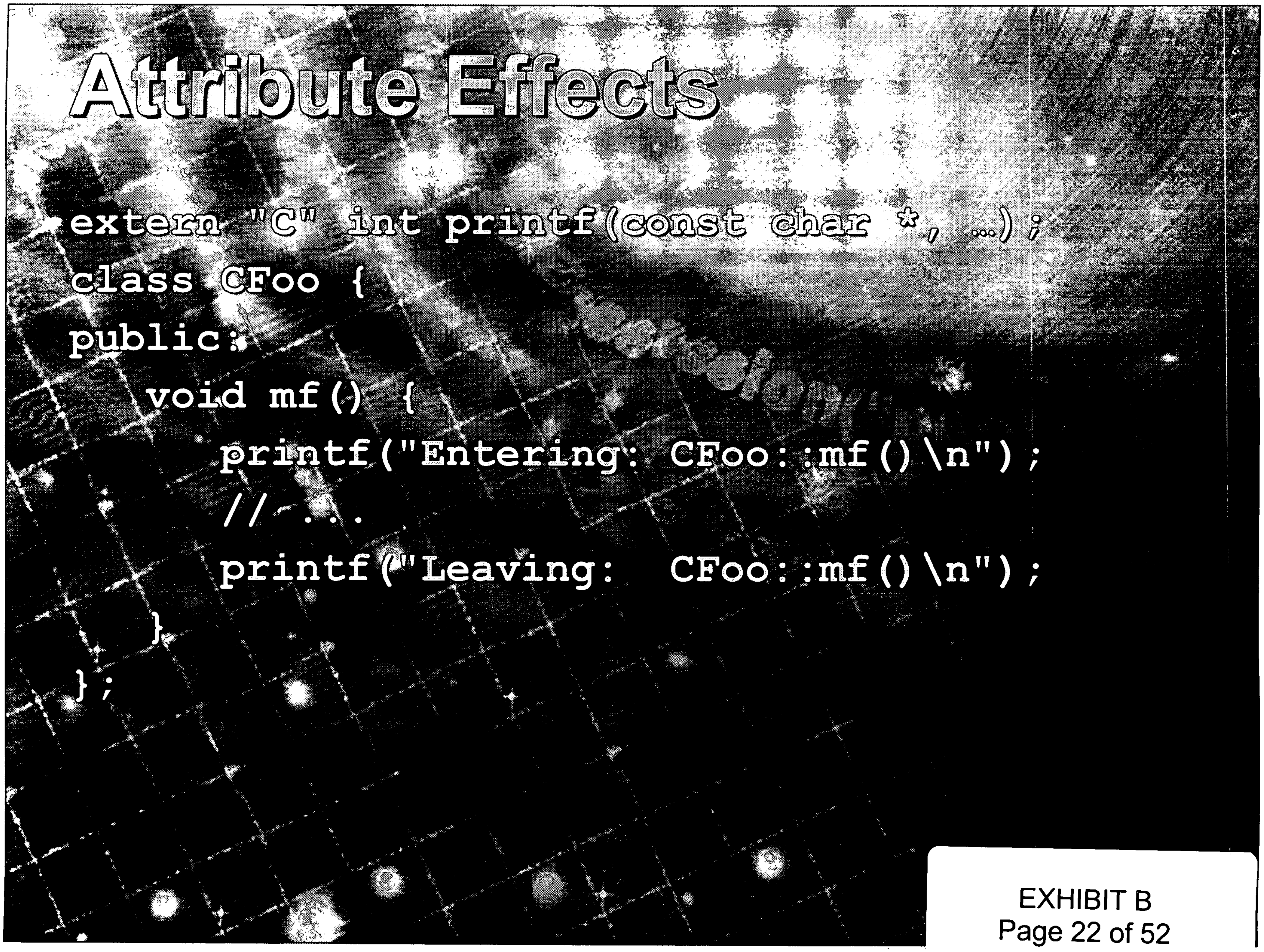
- Attributes can have many effects on a C++ program, these include:
  - Adding new classes or functions
  - Adding base-classes to a class
  - Adding methods to a class
  - Adding statements to a function



Attribute Effects

```
[ trace(1) ]
class CFoo {
public:
   void mf() {
      // ...
   }
};
```

EXHIBIT B
Page 21 of 52



Attribute Effects

```
extern "C" int printf(const char *, ...);
class CFoo {
public:
   void mf() {
       printf("Entering: CFoo::mf()\n");
       // ...
       printf("Leaving:  CFoo::mf()\n");
   }
};
```

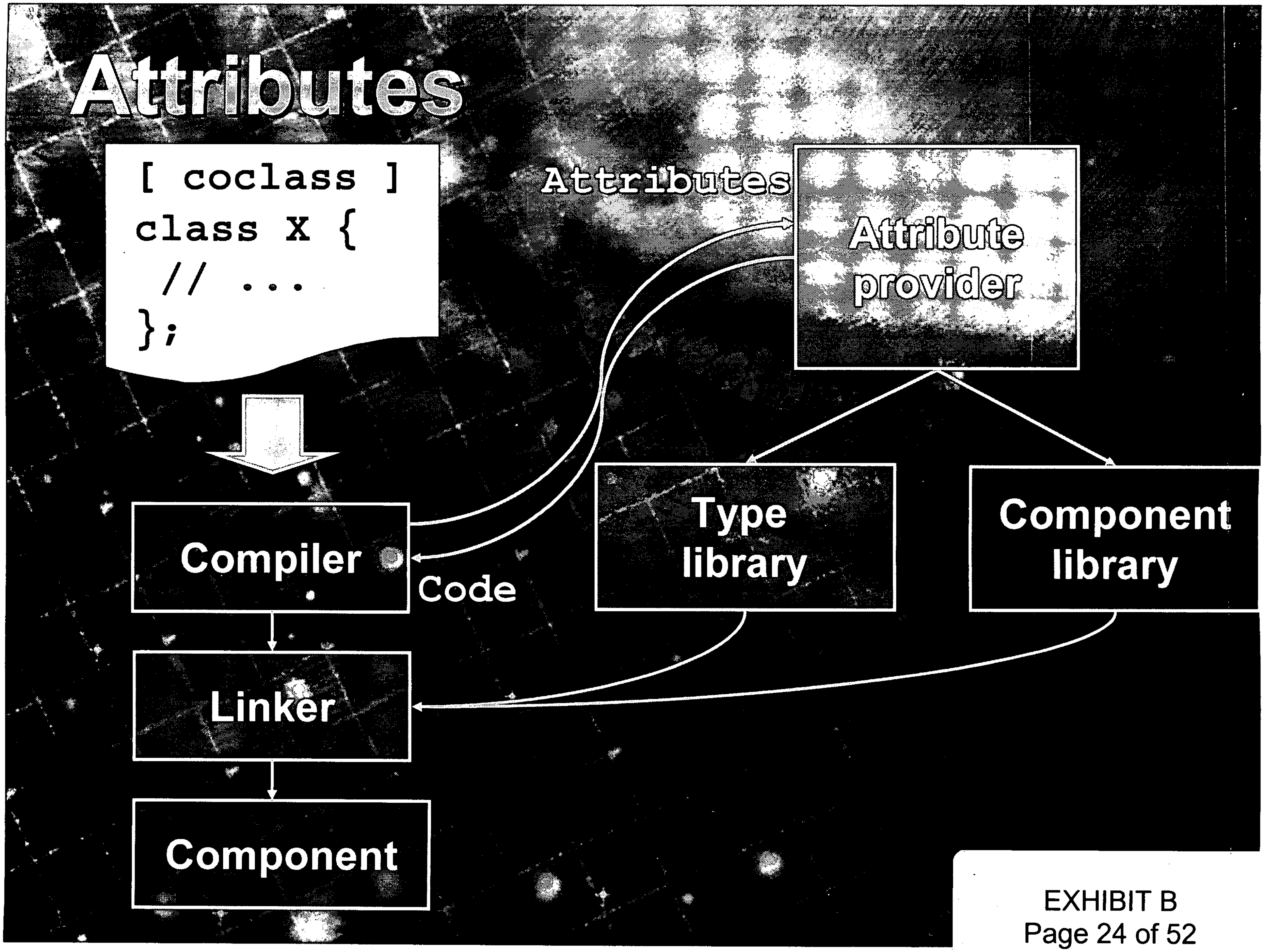
EXHIBIT B
Page 22 of 52

Attribute Effects

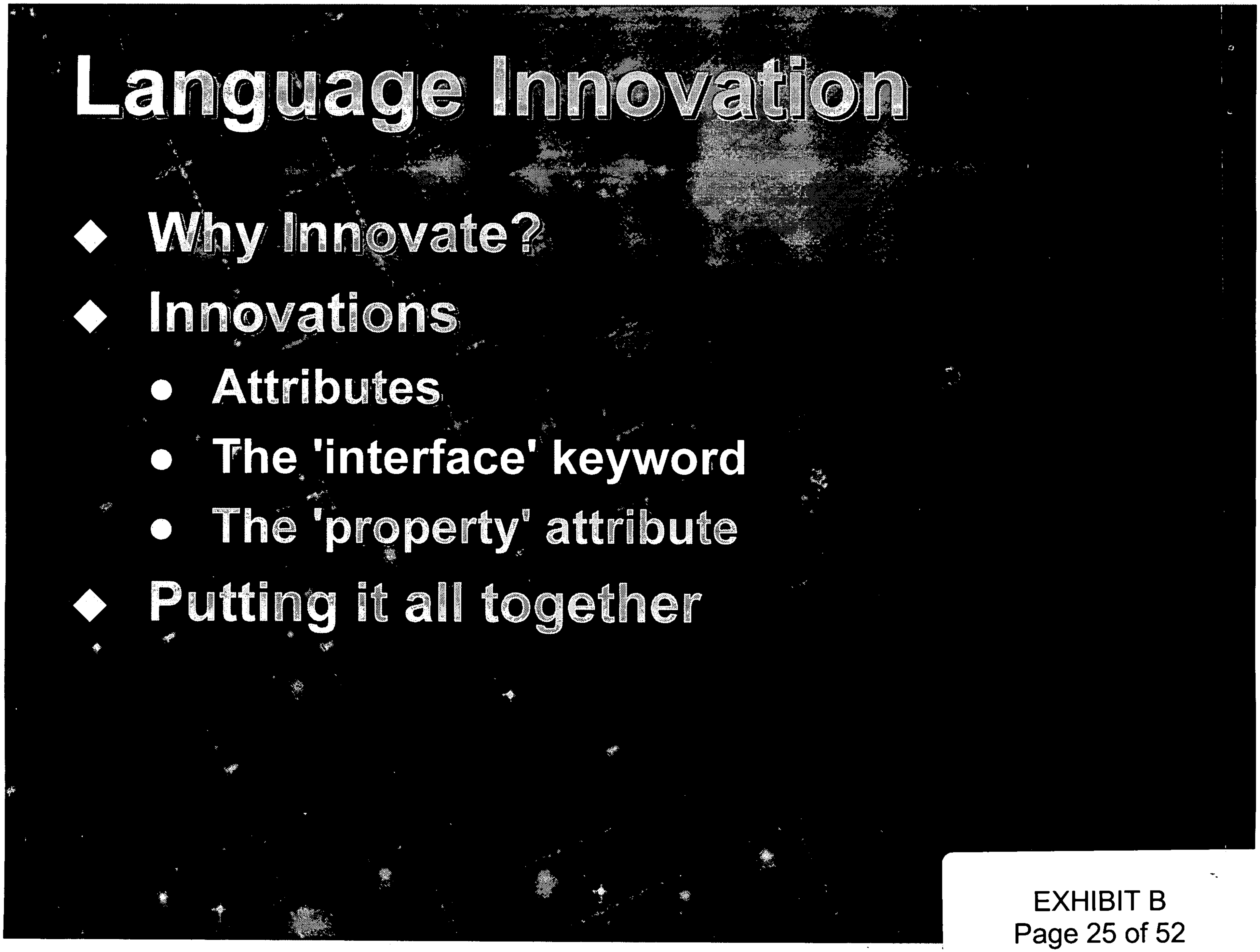
- Attributes can also cause effects unrelated to the normal C++ compilation process

```
[ com, uuid("..."), dual ]
interface IFoo {
  [id(1)] HRESULT m1([in] int val);
  [id(2)] HRESULT m2([out, retval] int *pval);
};
```

Attributes



```
[ coclass ]
class X {
 // ...
};
```



Language Innovation

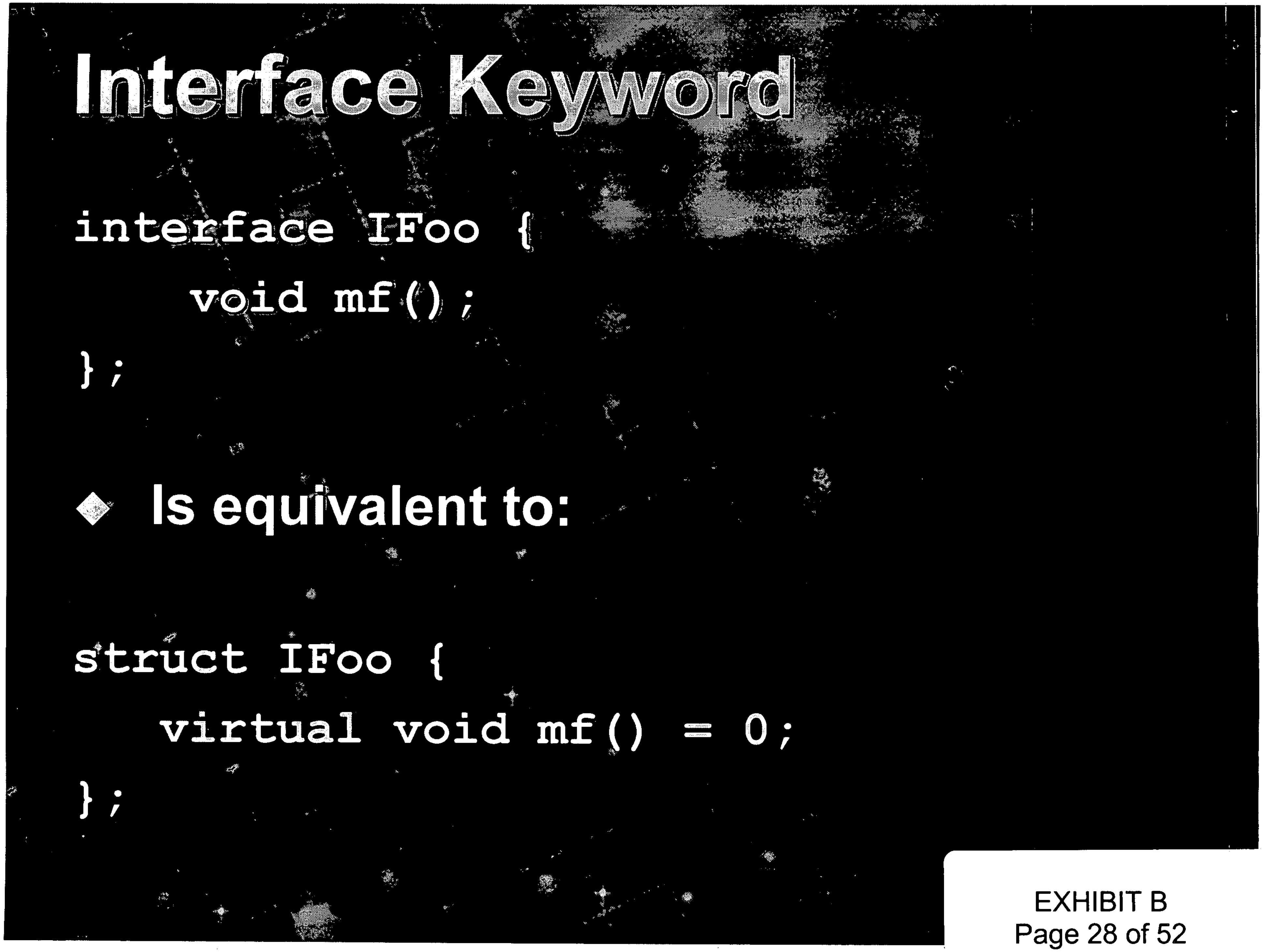
- Why Innovate?
- Innovations
  - Attributes
  - The 'interface' keyword
  - The 'property' attribute
- Putting it all together

Interface Keyword

- Can be thought of as a C++ struct with the following restrictions:
  - Can only inherit from another ‘interface’
  - Can only have methods
  - Methods must be public, pure virtual and can only be declared not defined

Interface Keyword

```
interface IFoo {
    void mf();
};
```



Interface Keyword

```
interface IFoo {
    void mf();
};
```

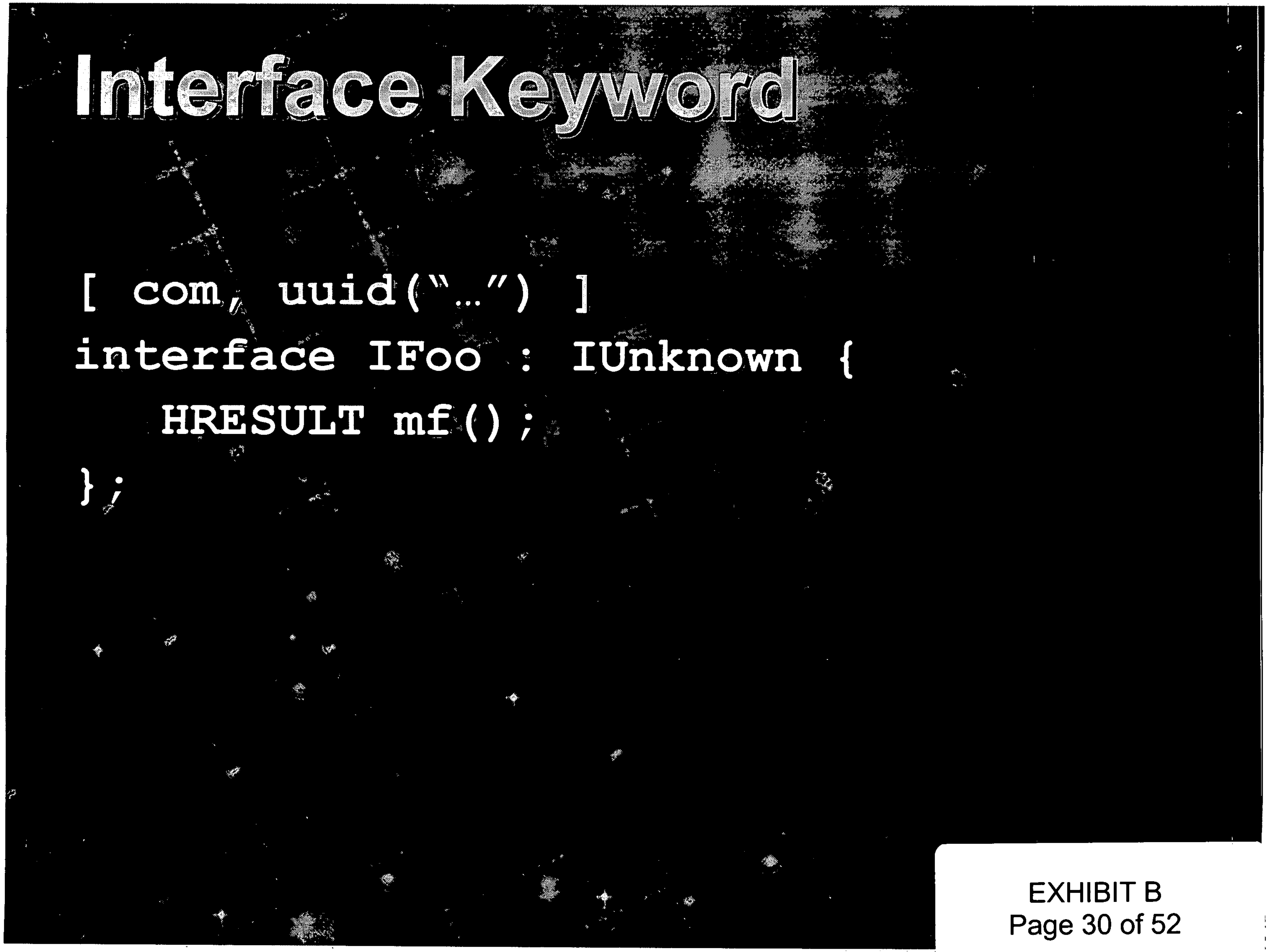
- Is equivalent to:

```
struct IFoo {
   virtual void mf() = 0;
};
```

Interface Keyword

- Applying the 'com' attribute to an interface makes it a COM interface imposes COM specific restrictions
  - Must have an IID
  - Must use a COM calling convention, normally __stdcall
  - Must inherit from another COM interface

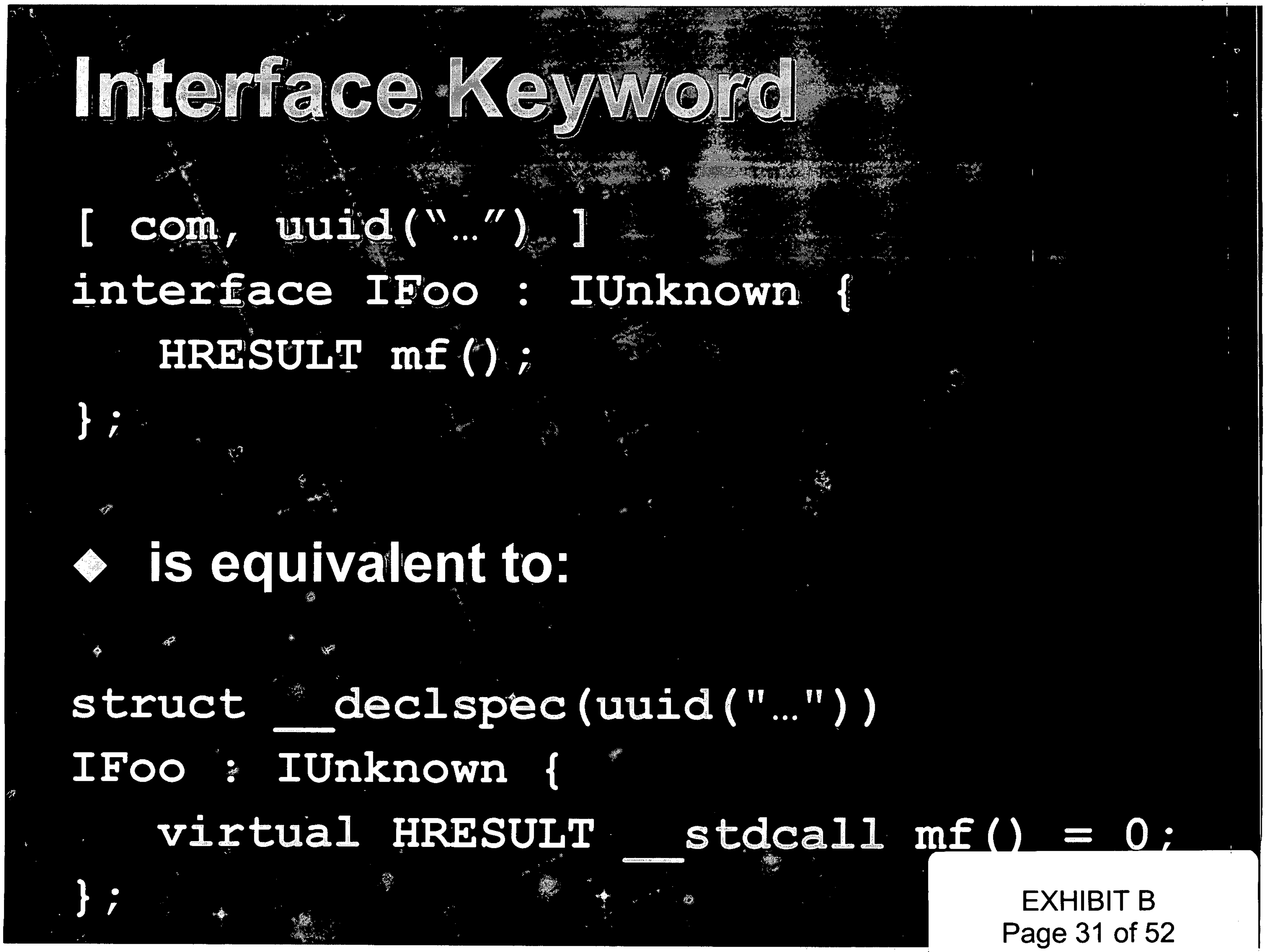
EXHIBIT B
Page 29 of 52



Interface Keyword

```
[ com, uuid("…") ]
interface IFoo : IUnknown {
   HRESULT mf();
};
```

EXHIBIT B
Page 30 of 52



Interface Keyword

```
[ com, uuid("…") ]
interface IFoo : IUnknown {
   HRESULT mf();
};
```

◆ is equivalent to:

```
struct __declspec(uuid("…"))
IFoo : IUnknown {
   virtual HRESULT __stdcall mf() = 0;
};
```

EXHIBIT B
Page 31 of 52

Interface Keyword

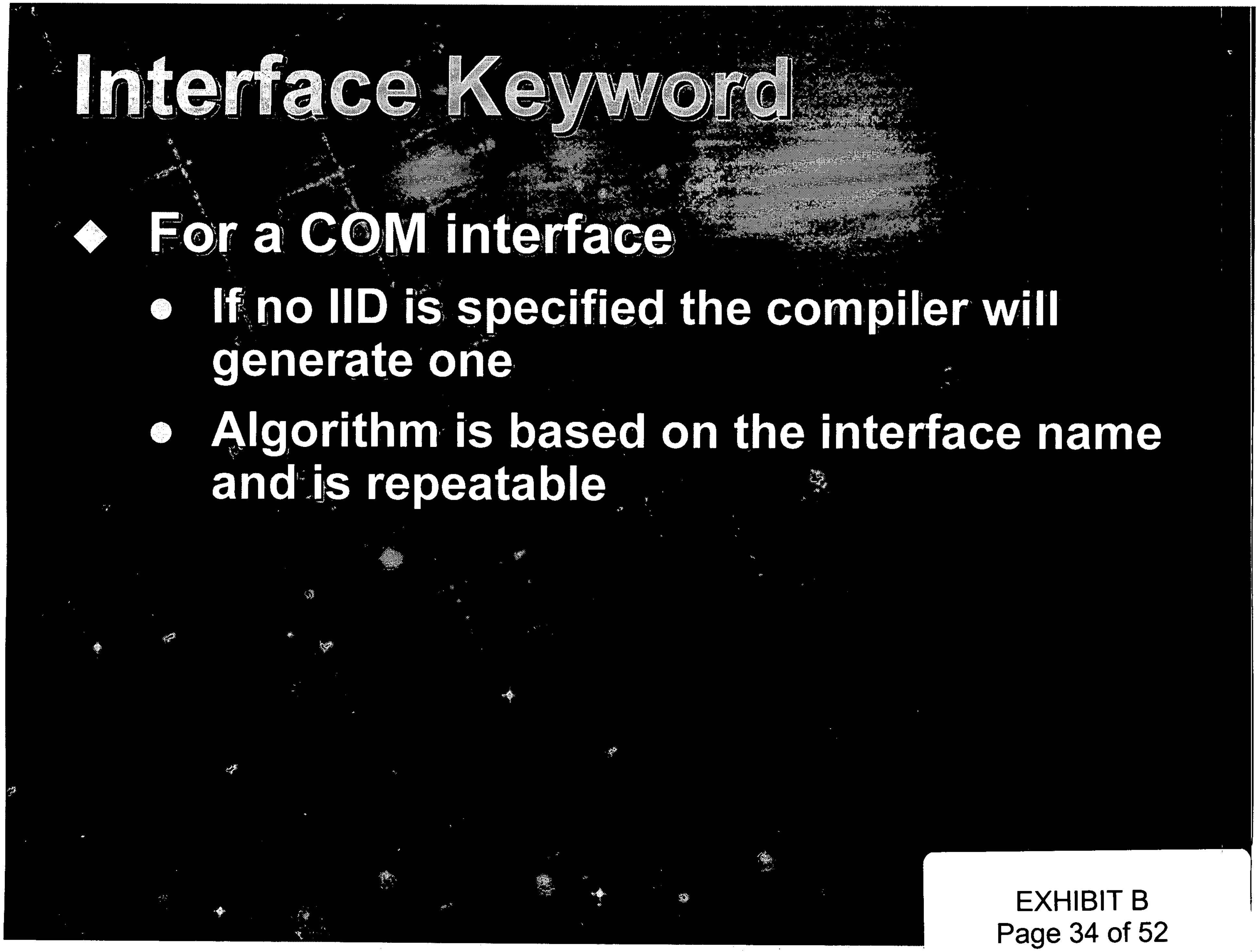
- A COM interface defaults to a 'custom' interface

```
[ com, uuid("…") ]
interface IFoo : IUnknown {
};

[ com, uuid("…") ]
interface IBar : IFoo {
};
```

Interface Keyword

- For a dual interface the ‘dual’ attribute must be used

```
[ com, uuid("…") ]
interface IFoo1 : IDispatch {
};

[ com, uuid("…"), dual ]
interface IFoo2 : IDispatch {
};
```



Interface Keyword

- For a COM interface
  - If no IID is specified the compiler will generate one
  - Algorithm is based on the interface name and is repeatable

Language Innovation

- Why Innovate?
- Innovations
  - Attributes
  - The 'interface' keyword
  - **The 'property' attribute**
- Putting it all together

Properties

- Can be thought of as 'virtual' data members
- Properties are implemented using attributes
- By default the compiler will generate a get and a put method for each property

Properties

- For a non COM interface

```
interface IFoo {
   [ property ]
   int Size;
};
```

Properties

- For a non COM interface

```
interface IFoo {
   [ property ]
   int Size;
};
```

- generates:

```
interface IFoo {
   const int & get_Size() const;
   void         put_Size(const int &);
};
```

Properties

- For a COM interface

```
[ com ] interface IFoo {
   [ property ]
   int Size;
};
```

Properties

- For a COM interface

```
[ com ] interface IFoo {
   [ property ]
   int Size;
};
```

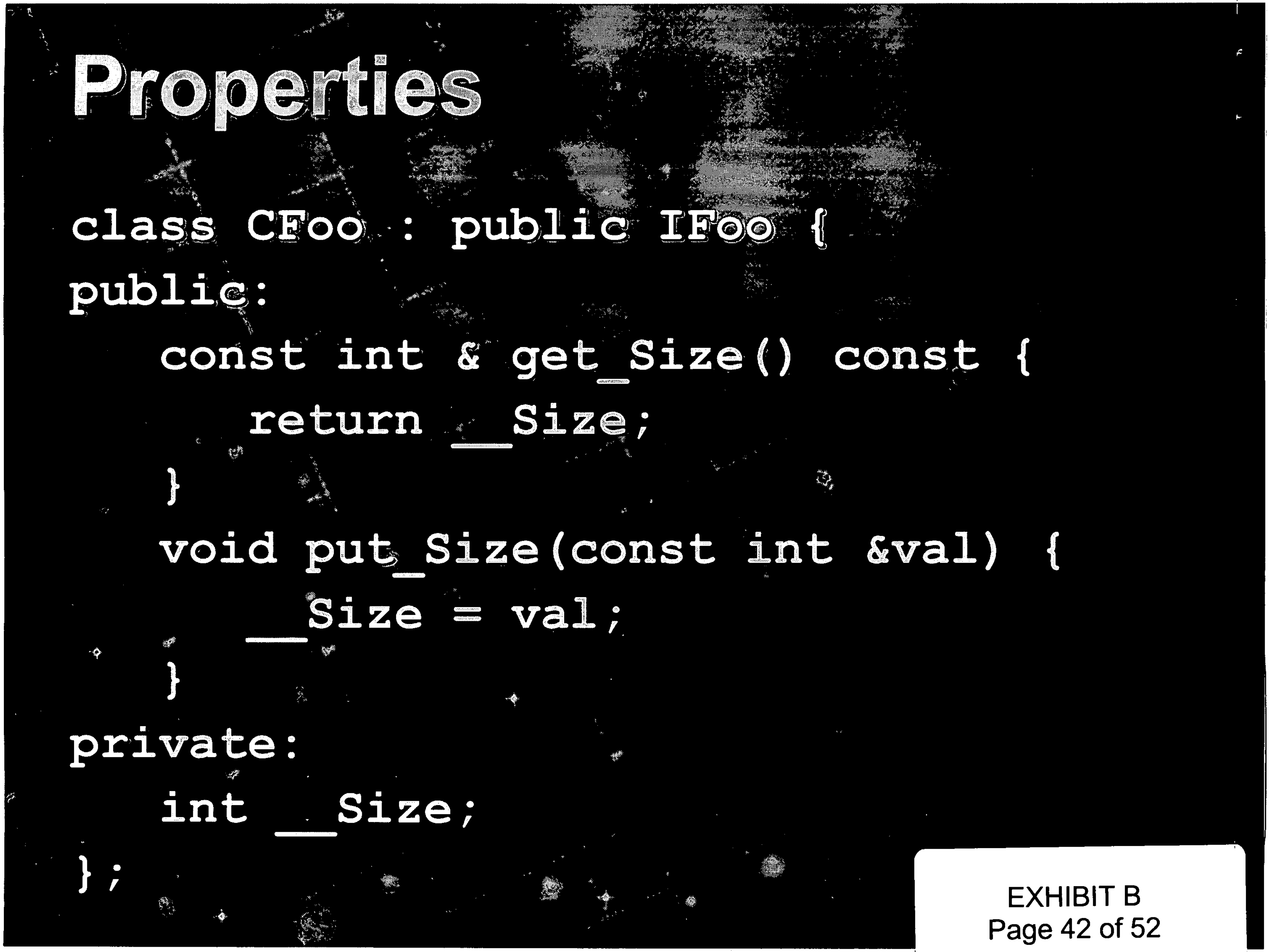
- Generates:

```
[ com ] interface IFoo {
   HRESULT get_Size([out, retval] int *);
   HRESULT put_Size([in] int);
};
```

Properties

- When used in a class or struct the compiler adds a data member and function definitions

```
class CFoo : public IFoo {
  [ property ]
  int Size;
};
```



Properties

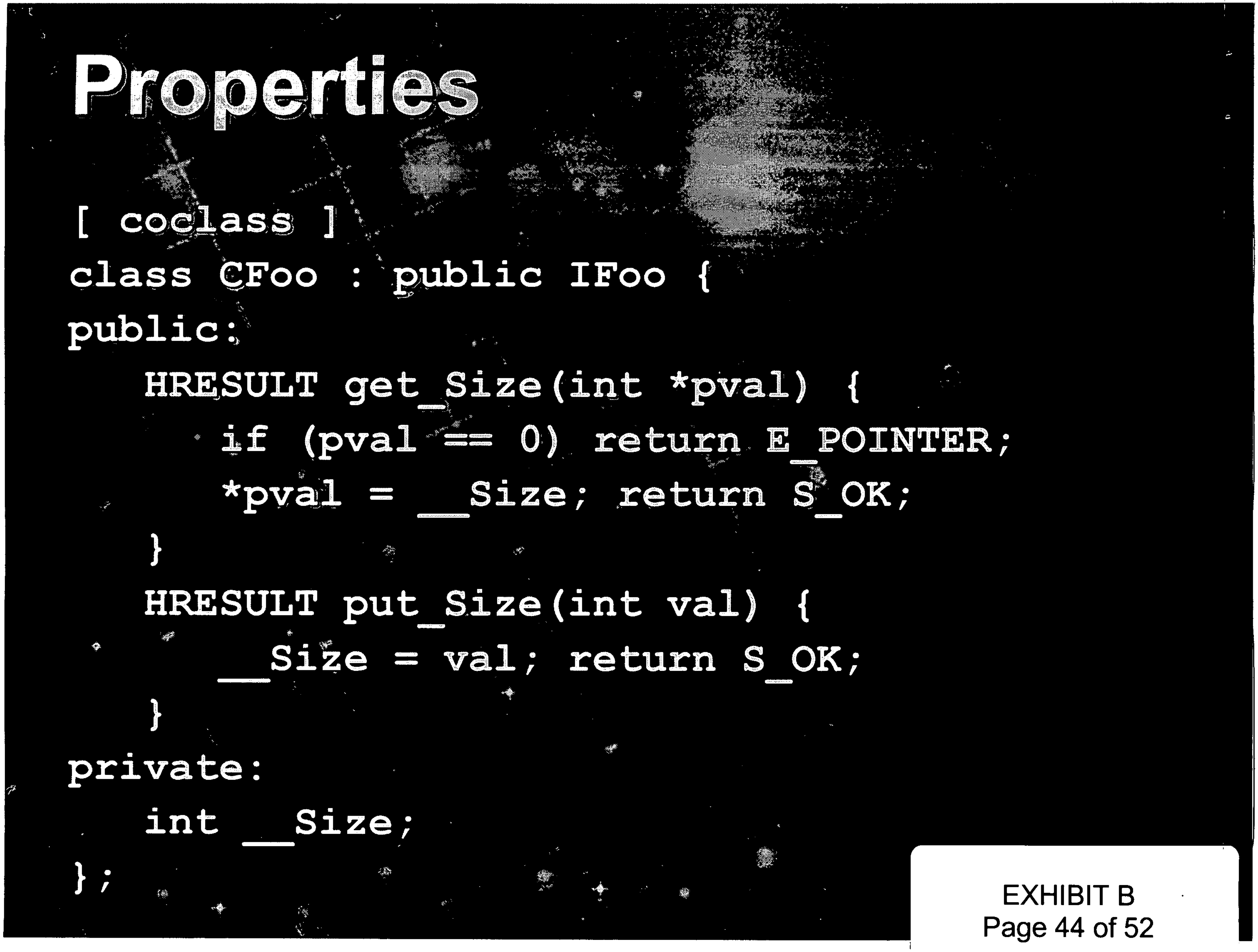
```
class CFoo : public IFoo {
public:
    const int & get_Size() const {
        return __Size;
    }
    void put_Size(const int &val) {
        __Size = val;
    }
private:
    int __Size;
};
```

EXHIBIT B
Page 42 of 52

Properties

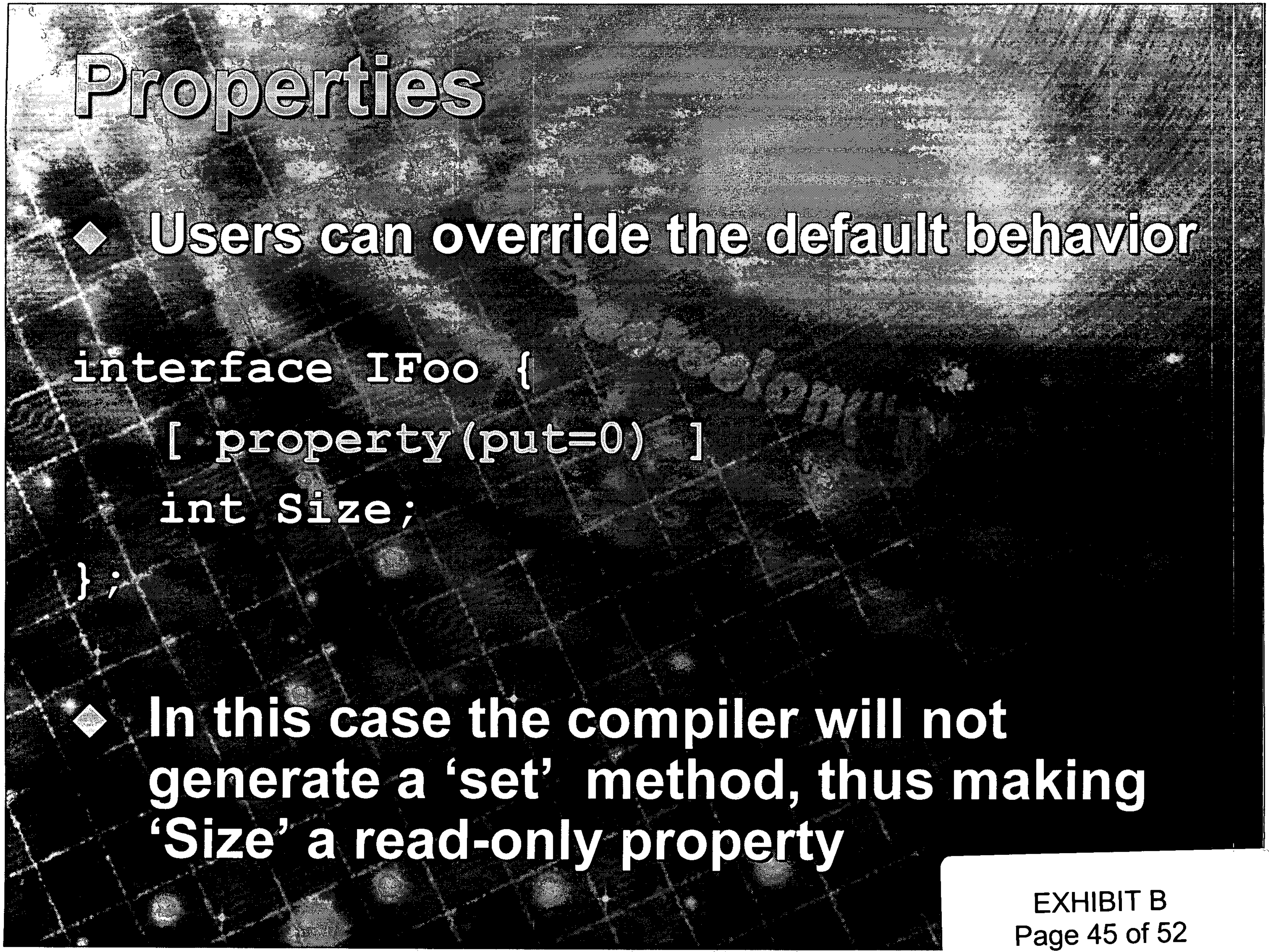
- For a COM class the functions are COM specific

```
[ coclass ]
class CFoo : public IFoo {
   [ property ]
   int Size;
};
```



Properties

```
[ coclass ]
class CFoo : public IFoo {
public:
   HRESULT get_Size(int *pval) {
      if (pval == 0) return E_POINTER;
      *pval = __Size; return S_OK;
   }
   HRESULT put_Size(int val) {
      __Size = val; return S_OK;
   }
private:
   int __Size;
};
```



Properties

- Users can override the default behavior

```
interface IFoo {
    [ property(put=0) ]
    int Size;
};
```

- In this case the compiler will not generate a ‘set’ method, thus making ‘Size’ a read-only property

EXHIBIT B
Page 45 of 52

Language Innovation

- Why Innovate?
- Innovations
  - Attributes
  - The 'interface' keyword
  - The 'property' attribute
- Putting it all together

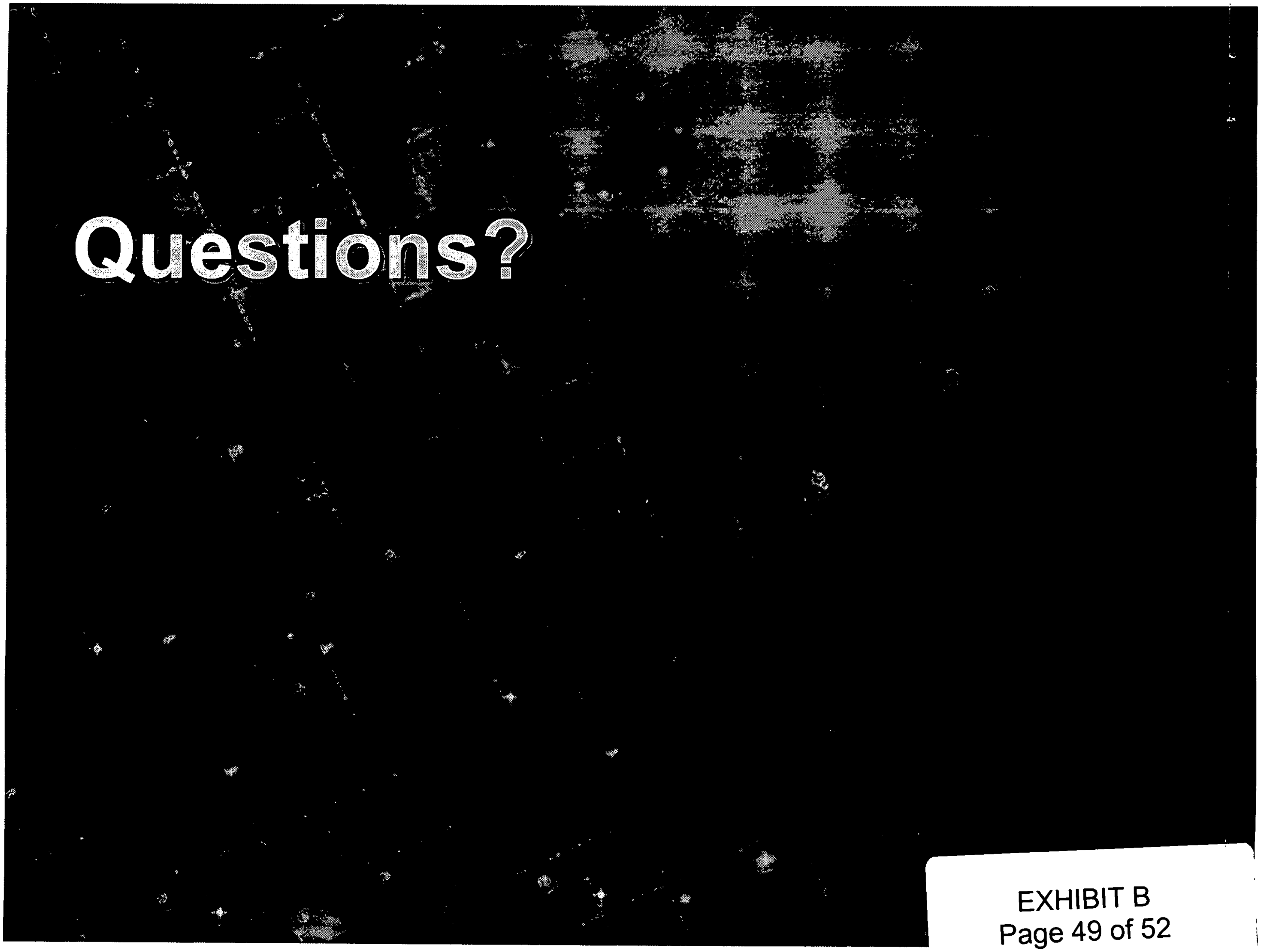
Putting It All Together

Demo

EXHIBIT B
Page 47 of 52

Call To Action

- Attributes make COM programming easier
- For new COM projects use all the Attributes
- For existing COM projects use the COM+ Attributes for easier registration



Questions?

EXHIBIT B
Page 49 of 52

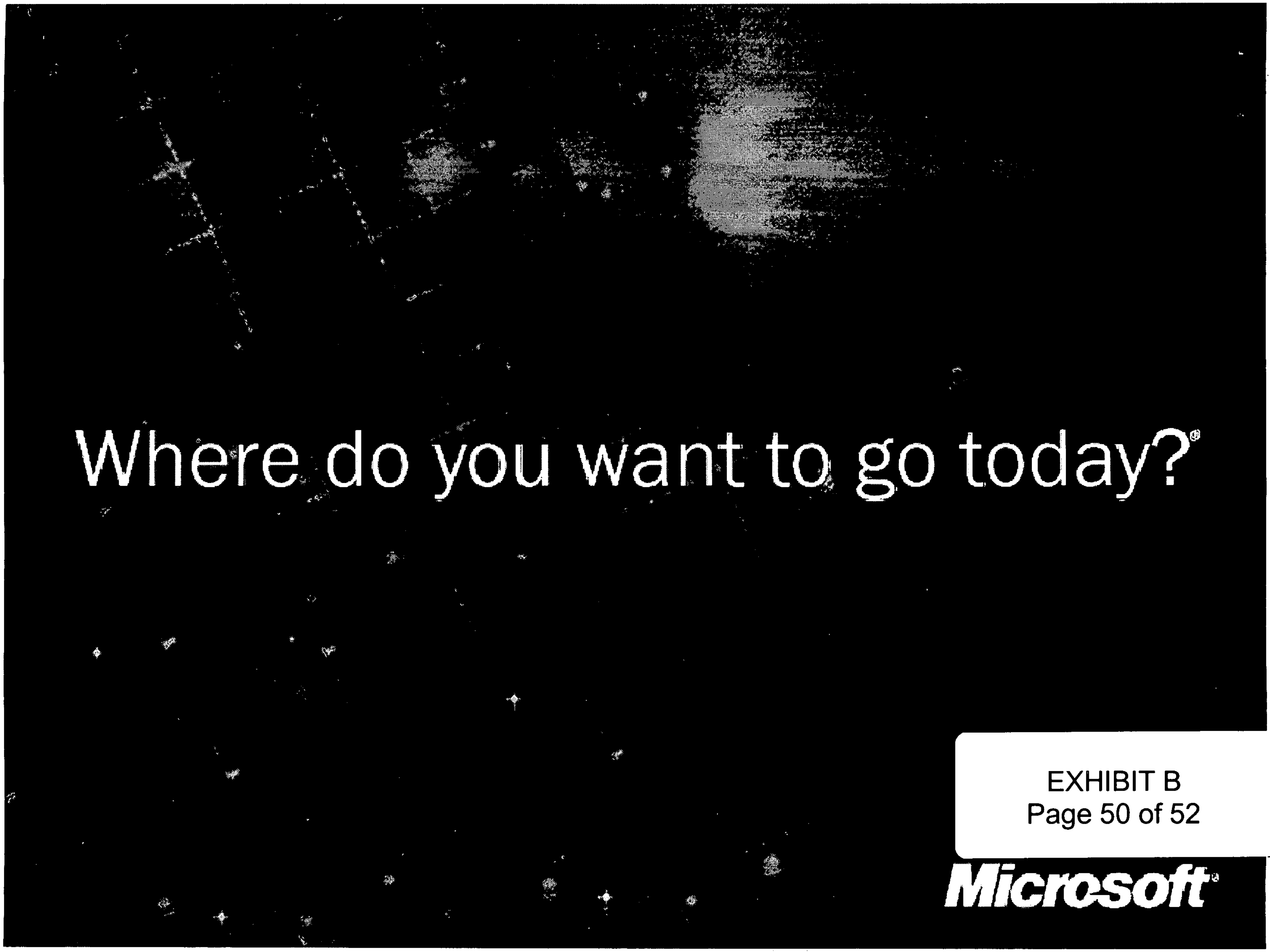


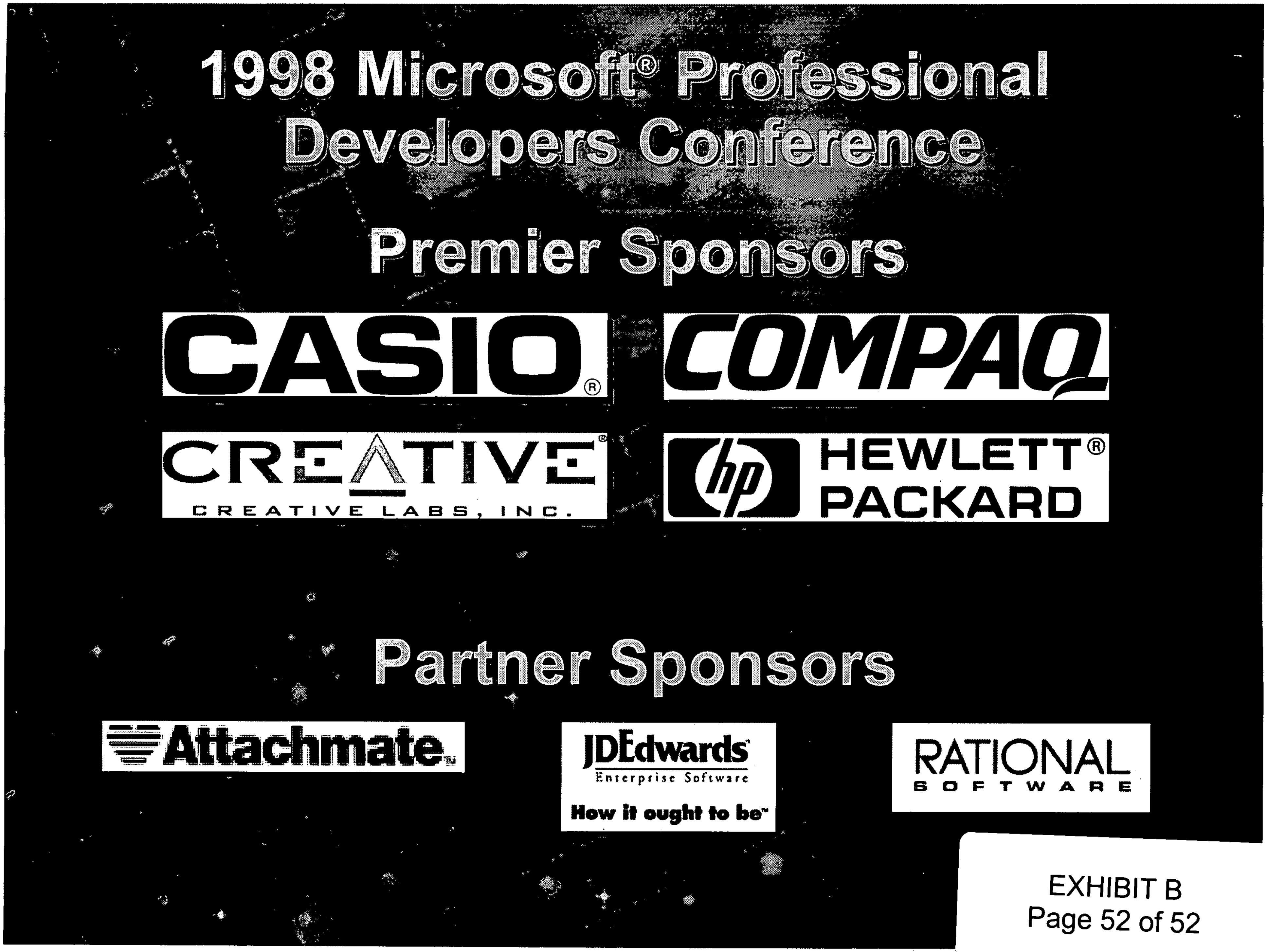
EXHIBIT B
Page 50 of 52

Building Windows-Based Applications for the Internet Age

Microsoft 1998 Professional Developers Conference

events.microsoft.com/events/pdc/

EXHIBIT B
Page 51 of 52



1998 Microsoft® Professional Developers Conference

Premier Sponsors

CASIO®

COMPAQ

CREATIVE® CREATIVE LABS, INC.

hp HEWLETT® PACKARD

Partner Sponsors

Attachmate™

JDEdwards® Enterprise Software How it ought to be™

RATIONAL SOFTWARE

EXHIBIT B
Page 52 of 52

EXHIBIT C

DECLARATION OF INVENTORS

We, Paul Ringseth, Jonathan Caves, and Jason Shirk, declare:

1. We are currently employed by the Microsoft Corporation.

2. In 1998 we worked as software design engineers on a software project which developed techniques and tools described at the conference talk entitled "Language Innovations for COM+ and Beyond" which was given at the 1998 Microsoft Professional Developers Conference in Denver, Colorado.

3. As designers for that project, we also directed preparation of a file entitled "Visual C++ Technical Preview" ("the Technical Preview"), which was distributed at that talk.

4. Major design decisions made in the course of production of the Technical Preview were reviewed by us.

5. The Technical Preview included a compiler which had a limited ability to compile code containing attributes, a single example project which could be compiled on the compiler, and some documentation.

6. While the example project was able to be compiled on the compiler, the compiler was intended for demonstration purposes only.

7. Because the compiler was included for demonstration purposes, it was not designed to be generally usable for compilation, but instead to demonstrate a specific example of compilation.

8. The compiler had not been tested generally on code other than the specific example project that was included with it.

9. The development team and I did not intend that the compiler be usable as a working general-purpose compiler and did not think of it as such.

10. The compiler was not, and was not intended to be, a finished commercial release.

11. The compiler was not, and was not intended to be, a beta version of a finished commercial release.

12. The compiler contained a bug which prevented executable files from being created.

13. The documentation included in the Technical Preview did not include information about general compilation. Instead, only enough information to compile the example project was included.

14. The example project was included to demonstrate software development using C++ code containing attributes.

15. The documentation did not describe the general use or capabilities of MIDL attributes which were able to be compiled by the compiler.

16. The documentation did not fully describe interfaces used by the compiler.

17. The compiler included in the Technical Preview comprised a C++ compiler, an MIDL compiler, and a text preprocessor.

18. The C++ compiler had no semantic knowledge of MIDL attributes.

19. The MIDL compiler had limited semantic knowledge of C++.

20. This means the C++ compiler could not compile MIDL code, nor could the MIDL compiler compile C++ code.

21. Because of this, the compiler included in the Technical Preview was only able to compile C++ containing attributes if the C++ code and MIDL attributes were separated at the start of compilation.

22. The text preprocessor included in the compiler performed the task of separating the C++ code and MIDL attributes.

23. The text preprocessor acted to "pass through" the MIDL attributes, and did not perform a semantic analysis on the MIDL attributes.

24. After separation, the C++ code was compiled by itself using the C++ compiler, and the MIDL attributes were compiled using the MIDL compiler.

25. After the separate compilation steps, the compiler combined the compiled C++ code and MIDL attributes into working executable code.

26. Because the C++ and MIDL compilers worked separately, information contained in the MIDL attributes was not available during C++ compilation.

27. The actions of the C++ compiler during compilation could not be affected by MIDL attributes being compiled by the MIDL compiler.

28. There was no intercommunication between the C++ compiler and the MIDL compiler.

29. The C++ compiler was not able to utilize MIDL attribute information either when creating internal representations of the C++ code or when generating machine-readable code.

30. The compiler (including the C++ compiler, MIDL compiler, and text preprocessor) could be configured to pass through MIDL attributes into a .obj file before final compilation without compiling the attributes.

We declare under penalty of perjury that the foregoing is true and correct. All statements made herein of our own knowledge are true and all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Executed this 23rd day of December, 2005, in Redmond, Washington.

Paul Ringseth

Paul Ringseth

JL

January 2006

Executed this 3rd day of ~~December, 2005,~~ in Redmond, Washington.

[illegible]

Jonathan Caves

Executed this 27th day of December, 2005, in Redmond, Washington.

Jason Shirk

Jason Shirk